(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 October 2002 (10.10.2002)

PCT

(10) International Publication Number
WO 02/080028 A1

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/GB02/01231

(22) International Filing Date: 15 March 2002 (15.03.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0108070.4 30 March 2001 (30.03.2001) GB

(71) Applicant *(for all designated States except US)*: **BRITISH TELECOMMUNICATIONS PUBLIC LIMITED COMPANY** [GB/GB]; 81 Newgate Street, London EC1A 7AJ (GB).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **JONES, Dean, Michael** [GB/GB]; 17A Stoke Street, Ipswich, Suffolk IP2 8BX (GB). **CUI, Zhan** [CN/GB]; 7 Squirrels Field, Colchester, Essex CO4 5YA (GB).

(74) Agent: **ROBINSON, Simon, Benjamin**; BT Group Legal Services, Intellectual Property Department, 8th floor, Holborn Centre, 120 Holborn, London EC1N 2TE (GB).

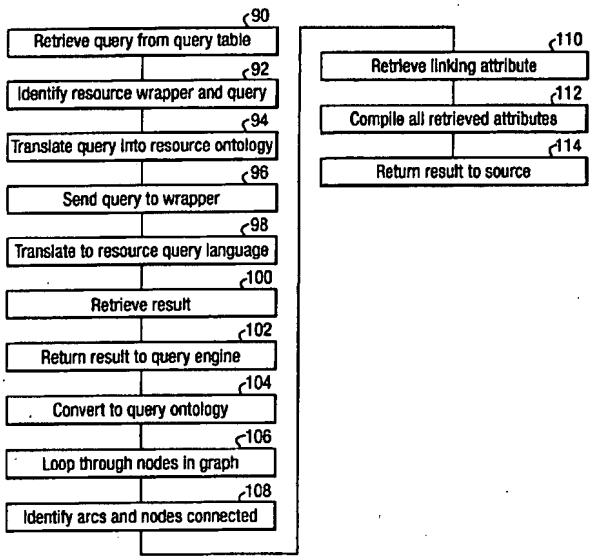
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Published:**
— *with international search report*

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

(54) Title: GLOBAL DATABASE MANAGEMENT SYSTEM INTEGRATING HETEROGENEOUS DATA RESOURCES




(57) Abstract: A database management system is disclosed for solving distributed queries across a range of resources. In known systems, database retrieval from multiple sources suffers from problems of reconciliation of data between resources and resource or data incompatibility. The invention allows full database integration even in the case where a database includes a plurality of disparate database resources having differing ontologies (data structures). The system has an ontology server which is used to store the ontologies of various database resources and the query engine is designed to consult the server so that it can construct a query that will operate effectively across the disparate database resources.

WO 02/080028 A1

# GLOBAL DATABASE MANAGEMENT SYSTEM INTERGRATING HETEROGENEOUS DATA RESOURCES

The invention relates to a database management system, in particular such a system for solving distributed queries across a range of resources.

In known systems, database retrieval from multiple sources suffers from problems of reconciliation of data between resources and resource or data incompatibility.

Aspects of the invention are set out in the attached claims.

The invention provides various advantages. In one aspect, the invention allows full database integration even in the case where a database includes a plurality of disparate database resources having differing ontologies.

In another aspect, the invention allows an integrated solution by finding and linking all database resources having the required elements for a specific database query.

In yet a further aspect, the invention allows a structured and efficient approach to solving a query by identifying sub-queries and dealing with each sub-query in turn or in parallel for integrating the sub-query results.

Embodiments of the invention will now be described, by way of example, with reference to the drawings, of which:

Fig. 1 is a block diagram of a network according to the present invention;

Fig. 2 is a block diagram of database resource schemas according to the present invention;

Fig. 3 is a block diagram of resource ontologies according to the present invention;

Fig. 4 is a block diagram of an application ontology according to the present invention;

Fig. 5 is a block diagram of a resource ontology-resource schema mapping according to the present invention;

Fig. 6 is a block diagram of an application ontology-resource ontology mapping according to the present invention;

Fig. 7 is a further block diagram of a network according to the present invention;

Fig. 8 is a flow diagram showing an initialisation sequence according to the present invention;

Fig. 9 is a node-arc representation of a concept identity graph according to the present invention;

Fig. 10 is a node-arc representation of a solution graph according to the present invention;

Fig. 11 is a node-arc diagram of an alternative solution graph according to the present invention; and

Fig. 12 is a flow diagram representing integration of data retrieved according to the present invention.

In overview, the invention provides a distributed query solution for a network having a plurality of database resources. In the preferred embodiment, the network used is a DOME network but it will be appreciated that any appropriate network can be used. The DOME network helps users to ask queries which retrieve and join data from more than one resource, such as an SQL or XML database.

When a query is received by the DOME query engine, it is treated as a request to retrieve values for a given set of attributes for all "individuals" that are instances of a given "concept" which also satisfy the given conditions. An "individual" is a specific field in a specific resource which may be duplicated, in another form, in another resource (e.g. in the specific example discussed below, two separate

database resources may have fields, under differing names, for a common entity such as a product name). A "concept" is in effect the query strategy – the query concept may be to retrieve all relevant product names for products satisfying given criteria, in which case the individuals are the fields in the resources carrying that information. The attributes are then the values (e.g. product names) associated with the relevant fields or individuals. The query engine constructs a set of sub-queries to send to the relevant resources in order to solve the user's query. Before the sub-queries are sent, the query engine will translate them into the vocabulary or "ontology" of the relevant resource. After the sub-queries are translated into the query language of the relevant resource (*e.g.* SQL) the results are passed back to the query engine. Once the query engine has received the results to all sub-queries, it will integrate them and pass the final results to the user client.

The subsequent discussion uses as an example a network 10 having three database resources 12, 14, 16, as illustrated in Fig. 1 comprising a "products" database 12, a "product prices" database 14 and a "product sales" database 16. The starting point for a DOME network is this set of resources. Although in principal any resource containing structured data can be included, here we discuss only relational databases. Examples of SQL resource schema for each of the resources in our running example are given in Fig 2, in which the schema for the products database is shown at 12a, for the product prices database at 14a and for the product sales database at 16a.

In setting up the network, first, a *resource ontology* is specified for each resource, which gives formal definitions of the terminology of each resource, ie database 12, 14, 16 connected to the network. Example resource ontologies are given in Fig. 3 for each of the products database 12b, products prices database 14b and product

sales database 16b. If the ontology of a resource is not available, it is constructed in order to make the meaning of the vocabulary of the resource explicit. For a database, for example, the ontology will define the meaning of the vocabulary of the conceptual schema. This ontology ensures that commonality between the different resources and the originating query will be available by defining the type of variable represented by each attribute in the schema. In addition, as shown in Fig. 4, an application ontology 18 is defined, providing equivalent information for the attributes required for a specific, pre-defined application, in the present case an application entitled "Product Analysis".

Having, by means of the ontology, effectively specified the data-type of each field or attribute in each of the distributed resources, a mapping is then specified between the resource ontology 12b, 14b, 16b and - in the case of a database - the resource schema 12a, 14a, 16a. This is shown in Fig. 5, for each of the products, product prices and product sales databases mappings 12c, 14c, 16c. Although it would be possible to define a mapping directly between an application ontology and the database schema, it is preferred to construct resource ontologies since the mapping between a resource ontology and a resource schema can then be utilised by different user groups using different application ontologies. This requires that relationships are also specified between an application ontology and a resource ontology before the query engine can utilise that resource in solving a query posed in that application ontology, as shown by mapping 18a in Fig. 6.

Fig. 7 shows the basic blocks of the network including the components described above. The resource ontologies 12b, 14b, 16b are stored in an ontology server 20, the resource ontology-application ontology mappings 18a are stored in a mapping server 22 and the resource schema-resource ontology mappings 12c, 14c, 16c are

stored in the relevant *wrapper* 24, 26, 28, which is an intermediary between the query engine 30 and a resource 12, 14, 16. A wrapper is responsible for translating queries sent by the query engine to the query language of the resource. In addition the network includes a query table 31, wrapper directory 32 and block 34 for the application ontology 18 as discussed in more detail below.

Once the various elements of the network have been started, the initialisation sequence begins as shown in Fig. 8. At step 40 each of the wrappers 24, 26, 28 registers with the directory 32 and lets it know at step 42 about the kinds of information that its respective resource 12,14,16 stores. In order to describe the information that is available in a resource 12, 14,16, a wrapper 24, 26, 28 needs to advertise the content of its associated resource with the directory 32. This is done in the terminology of the resource ontology 12b, 14b, 16b. This involves sending a translation into the resource ontology 12b, 14b, 16b of all possible parts of the resource schema 12a, 14a, 16a (*i.e.* those elements for which a resource ontology-resource schema mapping 12c, 14c, 16c has been defined.)

When the directory 32 receives an advertisement for an attribute of a resource 12, 14, 16, at step 46 it asks the ontology server if the role is an identity attribute for the concept (ie is the attribute listed in the application ontology 18) and the role is marked accordingly in the directory 32 database. Once each wrapper 24, 26, 28 has been initialised, the directory 32 is then aware of all resources 12, 14, 16 that are available and all of the information that they can provide. When a resource 12, 14 16 becomes unavailable (for whatever reason), at step 48 the wrapper 24, 26, 28 will communicate this to the directory 32 which updates at step 50 such that the information stored in the resource 24, 26, 28 will no longer be used by the query engine 30 in query solving.

A detailed description of the ontology translation techniques used in DOME is not necessary as the relevant approach will be well known or apparent to the skilled person. However an outline is provided that is sufficient for giving the detail of how a query plan is formed. In order to allow the translation of expressions from the vocabulary of one ontology to that of another, a set of *correspondences* are specified between the vocabularies of two ontologies. A correspondence between two concepts contains principally: the name of the source and target ontology and the source and target concept names. In some cases the correspondence also contains any pre- and post-conditions for the translation which are important for ensuring that the translation of an expression into the vocabulary of a target ontology has the same meaning as the original expression in the vocabulary of the source ontology. However this last aspect is not relevant to the present example.

The next step is to specify the elements that will be used when the query engine processes queries. In the preferred embodiment an object-oriented framework is used and so the methods associated with each element are also outlined.

A query that is passed to the query engine 30 has the following components: the ontology in which the terms used in the query are defined; a concept name; a set of names of attributes of the query concept for which values should be returned to the user client; a set of attribute conditions; and a set of role conditions. An attribute condition is a triple $\langle an, op, val \rangle$ where *an* is the name of an attribute of the query concept, *op* is an operator supported by the query language (*e.g.* '<', '>', '=' and so on) and *val* is a permissible value for the given attribute or operator. In the specific example described herein are the names of the attributes in each of the conditions is relevant. Each of the role conditions is also a triple $\langle rn, op, sq \rangle$

where *rn* is the name of a role, *op* is an operator (*e.g.* 'all', 'some') and *sq* is a sub-query. The sub-query itself largely conforms to the above guidelines for queries but does not specify the name of the ontology, since this will be the same (it being a sub-set of the main query), or the names of attributes for which values should be returned, since these will be determined automatically. In the specific example discussed herein the operators in role conditions are not relevant.

In the specific example scenario, the user wants to find the name and code of all products which are made by companies with more than 100 employees and which have sold more than 10,000 units. We can represent this query more formally as:

```
(Product-Analysis-Ontology, Product,
{Product.product-name,Product.product-code}
{Product.product-sales}
{Product.manufacturer,(Manufacturer,{Manufacturer.emplo
yees},{})
)
```

where the application concept is "Product Analysis", the attributes or individuals in the application are product name, code and sales and manufacturer employees and the resources are the product, product prices and product sales databases 12, 14, 16.

When describing the algorithms used in query processing, it is assumed objects exist that belong to the following classes (with associated methods):

*Query* - represents a query sent to the query engine

*Query(c, o)* - a constructor method which takes a concept name and an ontology name as arguments

*getOntology()* - returns the name of the ontology in which the query is framed

*getConcept()* - returns the name of the query concept

*getRequiredAttributes()* - returns the set of required attributes

*getAttributeConditions()* - returns the set of attribute conditions

*getRoleConditions()* - returns the set of role conditions

*addRequiredAttribute(a)* - adds *a* to the set of required attributes

*addAttributeCondition(ac)* - adds *ac* to the set of attribute conditions

*addRoleCondition(rc)* - adds *rc* to the set of role conditions

*RoleCondition*

*getRole()* - returns the role in the condition

*getSubQuery()* - returns the condition's sub-query

*setSubQuery()* - sets the value of the sub-query part (note that during processing, this can be set to the results to the sub-query)

*AttributeCondition*

*getAttribute()* - returns the attribute in the condition

*QueryEngine*

*askQuery(q)* - the response to a query will be a table of values where each column corresponds to the values for an attribute and each row corresponds to the values for an individual

*Directory*

*knows(c, o)* - returns the set of wrappers that know about the concept c defined in ontology o

*knows(a, c, o)* - returns the set of wrappers that know about the attribute a of the concept c defined in ontology o

*Wrapper*

*askQuery(q)* - retrieve the results to the query q from the wrapper's associated resource

*knows(c, o)* - returns true if the wrapper knows about the concept c defined in the ontology o

*knows(a, c, o)* - returns true if the wrapper knows about the attribute a of the concept c defined in the ontology o

*getPrimaryKey(c, o)* - retrieve the key attribute(s) for concept c in ontology o

Accordingly commands are defined allowing operation of the query engine as discussed below.

When the query engine receives a query, a plan is constructed to solve the query given the available information resources and the algorithm for constructing such a plan is discussed below. Queries are solved recursively. The query engine first tries to solve each member of the set of sub-queries. Any of these that do not themselves have complex sub-queries can be solved directly (if the required information is available).

A number of different data structures are utilised in the following description. In order to keep the description as generic as possible, it is assumed that these data structures are implemented as objects, referring to the following objects and methods:

*Graph* - represents a graph consisting of a set of nodes and a set of arcs
*addNode(n)* - add node *n* to the graph
*addArc(m, n, l)* - add an arc between nodes *n* and *m* with the label *l*
*removeNode(n)* - remove the node *n* from the graph
*connected()* - return true if the graph is connected
*getNodes()* - returns the set of nodes
*getSubGraphs()* - return the set of connected sub-graphs of the graph

*Hashtable* - a table of keys and associated values
*put(k, v)* - associate the key *k* with the value *v* in the table
*get(k)* - returns the value associated with the key *k*
*hasKey(k)* - returns true if the hashtable contains an entry with the key *k*

Accordingly the relevant structures are defined for subsequent processing of the query.

The next stage is to construct a "Concept Identity Graph" designated generally 60 as shown in Fig. 9, a directory and resources with wrappers having been established. The concept identity graph 60 represents, by linking them, the resources (ie databases 12, 14,16) via the respective wrappers 24, 26, 28 that have the same primary key attribute (or attributes for composite keys) for a concept. Given some query *q*, a concept identity graph for the query concept defined in

some ontology is constructed using the following algorithm, based on the commands and data structures discussed above:

**Inputs:** *query* - the query

**Output:** *graph* - the concept identity graph for *query*

```
initialise graph
o = query.getOntology()
c = query.getConcept()
wrappers[] = directory.knows(c, o)
for i = 0...|wrappers|-1
    graph.addNode(wrappers[i])
    primaryKey = wrappers[i].getPrimaryKey(c, o)
for j = 0...i-1
if primaryKey = wrappers[j].getPrimaryKey(c, o) then
            g.addArc(w[i], w[j], primaryKey)
return g
```

In solving the top-level query in our example, the graph 60 in Fig. 9 is constructed. The wrappers related to resources having the relevant fields or attributes are identified and created as nodes. An arc 62 between nodes is created when the nodes so linked share a key attribute, ie, an attribute demanded by the query. Where there is an arc 62 between a pair of wrappers 24, 26, 28 in the graph 60, we can directly integrate information about the query concept that is retrieved from the resources 12, 14, 16 associated with those wrappers. In the example, information about products which is retrieved from the Product-Price resource 14 can be integrated with information about products retrieved from either the

Products resource 12 or the Product-Sales resource 16, but information about products retrieved from the Products and Product-Sales resource cannot *directly* be integrated as there is no linking arc 62. For this reason, in order to ensure that information from two resources can be integrated, they must at least be in the same sub-graph of the concept identity graph 60, where a sub-graph may be the only graph or one set up to accommodate a sub-query forming part of an overall query (how information retrieved from two resources that are not neighbours in the concept identity graph may be integrated *indirectly* is discussed below).

The next stage is to construct queries to send to resources. The user query can be solved if it is ensured that:

(a) each condition and each user-specified required attribute is allocated to at least one resource query and

(b) the results to the resource queries can be integrated.

In other words, all the information required is available from one or other of the resources, and the resources are not themselves incompatible such the that information cannot be collated.

Starting with requirement (a), attribute conditions and required attributes can be allocated simply to resource queries by identifying resources that contain the relevant attributes. A slight complication is that, as outlined above, those resources must be in the same connected sub-graph of the concept identity graph 62, which is ensured by selecting one sub-graph at a time.

In the top-level query, the user specifies the attributes for which values should be returned. For sub-queries embedded in role conditions, this is not the case. There, the attributes for which values from the resource must be retrieved (e.g. for a

query to an SQL database, which fields are named between 'SELECT' and 'FROM') must be determined. This is done by first finding a resource that answers the role condition and using this to determine the values that need to be retrieved. The system loops through the relevant resources until one can be found which allows the sub-query to be solved. The results to the sub-query are retrieved by issuing a query against the query engine (demonstrating again how queries are solved recursively) and the sub-query in the role condition is then replaced with these results.

The following algorithm demonstrates how required attributes and conditions are allocated to resource queries.

**Inputs:** *q* - the user query
*g* - the concept identity graph for the query concept

**Output:** *resourceQueryHashtable* - mapping of wrappers to resource queries
*subGraph* - the component of the concept identity graph that enabled all parts of the query to be allocated

```
subGraphs[] = g.getSubGraphs()
o = q.getOntology()
c = q.getConcept()
requiredAttributes[] = q.getRequiredAttributes()
attributeConditions[] = q.getAttributeConditions()
roleConditions[] = q.getRoleConditions()
allAllocated = false
for i = 0...|subGraphs|
subGraphNodes[] = subGraphs.getNodes()
// allocate the user-specified required attributes
for j = 0...|requiredAttributes|-1
    for k = 0...|subGraphNodes|-1
    if subGraphNodes[k].knows(requiredAttributes[j], c, o) then
    if resourceQueryHashtable.hasKey(subGraphNodes[k])
                resourceQuery                                    =
            resourceQueryHashtable.get(subGraphNodes[k])
```

```
                    resourceQuery.addRequiredAttribute(requiredAttributes[j])
                else
                resourceQuery = new Query(c, o)
                resourceQuery.addRequiredAttribute(requiredAttributes[j])
                resourceQueryHashtable.put(subGraphNodes[k], resourceQuery)
// allocate the attribute conditions
    for j = 0... |attributeConditions|-1
        for k = 0 ... |subGraphNodes|-1
    if subGraphNodes[k].knows(attributeConditions[j].getAttribute(), c, o) then
    if resourceQueryHashtable.hasKey(subGraphNodes[k])
                    resourceQuery                                          =
                resourceQueryHashtable.get(subGraphNodes[k])
                    resourceQuery.addAttributeCondontion(attributeConditions[j])
                else
                resourceQuery = new Query(c, o)
                resourceQuery.addAttributeCondition(attributeConditions[j])
                resourceQueryHashtable.put(subGraphNodes[k], resourceQuery)
    // allocate the role conditions
    for j = 0...|roleConditions|-1
    subQuery = roleConditions[j].getSubQuery()
    for k = 0...|subGraphNodes|-1
    if subGraphNodes[k].knows(roleConditions[j].getRole(), c, o) then
primaryKey = resources[k].getPrimaryKey(subQueryConcept, o)
            subQuery.addRequiredAttribute(primaryKey)
subQueryResults = queryEngine.askQuery(subQuery)
roleConditions[j].setSubQuery(subQueryResults)
    if resourceQueryHashtable.hasKey(subGraphNodes[k])
                    resourceQuery                                          =
                resourceQueryHashtable.get(subGraphNodes[k])
                    resourceQuery.addRoleCondition(roleConditions[j])
                else
                resourceQuery = new Query(c, o)
                resourceQuery.addRoleCondition(roleConditions[j])
                resourceQueryHashtable.put(subGraphNodes[k], resourceQuery)
    if allAllocated = true
        return resourceQueryHashtable and subGraph[i]
```

Accordingly the algorithm allocates attributes, attribute conditions and role conditions by assessing the contents of the subgraph node resources. If some user-

specified required attribute or condition cannot be allocated to a resource query, the user query cannot be solved by the current set of resources connected to the network and the user is informed.

Having shown how conditions and required attributes are allocated to resource queries, the next stage is ensuring that the results to these resource queries can be integrated. The connected sub-graph for which all of the required attributes and conditions can be allocated to a resource query is termed the *solution graph* 70 in Fig. 10. If some part of the user query has been allocated to a resource 12, 14, 16, we say that the resource is *active* in relation to a given query. The next stage is to ensure that it will be possible to integrate the results to each of the resource queries. In order to be able to integrate the results from two active resources (designated in the figure by the respective wrapper 24, 26, 28) which are neighbours in the solution graph 70, we need to retrieve values for an identity attribute 72a,b which labels the arc 62 joining the resources. It follows that if all of the active resources are neighbours in the solution graph 72, that is to say, they are linked by an arc 62 designating a shared attribute, provided we retrieve values for the correct attributes, we can integrate the results to all of the resource queries. For example, if there is a solution graph as shown in Fig. 10 with the active resources 24, 26 being shown as solid nodes, in order to integrate results to the two resource queries, it is necessary to retrieve the data for 'product-name' from each resource.

However, if an active resource does not have any active neighbours in the solution graph, it will not be possible to integrate the results from the corresponding resource query without some additional information. The solution adopted to this problem is to construct a set of one or more *intermediate queries* which are sent to the resources to retrieve data that is then used to integrate the results of the

resource queries. An intermediate query 6b must be sent to each resource that lies on the path between (a) the active resource without any active neighbours, and (b) the nearest active resource to it. For example, consider the solution graph shown in Fig. 11. In order to integrate data from the active resources product and product sales 12, 16 represented by solid nodes an intermediate query 80 is sent to the 'Product-Price' resource 14 which retrieves information on the 'product-name' and the 'product-code' attributes. If we the 'product-name' data is retrieved from the 'Products' resource 12 and the 'product-code' data from the Product-Sales resource 16, the results can be used at the intermediate query 80 to integrate the result from the two resource queries. It may be that in order to make a path between two nodes that are active in a query, multiple intermediate queries are required dependent on the complexity of the query.

The algorithm to determine whether any intermediate queries are required is shown below and is based on determining whether the sub-graph that contains the active nodes is connected. If so, a solution has been found. If not, additional nodes are added until the graph is connected. Nodes are added by generating a combinations of inactive nodes, adding these to the graph and then determining whether the resulting graph is connected. Combinations of increasing length are generated *i.e.* if there are $n$ inactive nodes in the graph, combinations are generated in order combinations of lengths 1 up to $n$. Combinations can be generated using an implementation of one of the many known algorithms generating combinations, for example Kurtzberg's Algorithm (Kurtzberg, J. (1962) "ACM Algorithm 94: Combination", *Communications of the ACM* 5(6), 344).

This algorithm is implemented as below, taking the implementation as the function *Combination*($n$, $i$) where $n$ is the cardinality of the set of numbers to choose from

and *i* is the length of the combinations to generate. This function returns a set of all possible combinations of length *i*.

**Inputs:** *subGraph* - the component of the concept identity graph that enabled all parts of the query to be allocated

*resourceHashtable* - the mapping of wrappers to resource queries

```
// determine which nodes are active
solutionGraph = subGraph
solutionGraph[] = subGraph.getNodes()
for i = 0...|subGraphNodes|-1
if not resourceGraph.hasKey(subGraphNodes[i])
inactiveNodes[].add(subGraphNodes[i])
subGraph.removeNode(subGraphNodes[i])
if not subGraph.connected() then
foundSolution = false
i = 0
    while i <= |inactiveNodes| and not foundSolution
allCombinations[] = Combination(inactiveNodes, i)

for j = 0...|allCombinations|-1
combination[] = allCombinations[j]
for k = 0...|combination|-1
subGraph.addNode(combination[k])
// also need to add the arcs
```

add each resource in the combination as a node in the solution graph

add the required arcs to the solution graph

if (solutionGraph.isConnected

*foundSolution* := true;

else

remove nodes and arcs from the graph.

The final stage is to retrieve and integrate the data, and the system is illustrated with reference to Figs. 7 and 12. In order to send the resource queries, at step 90 the system loops through the *resourceQueryTable* 31 and retrieves at step 92 each entry in turn, which will consist of the identity of a resource wrapper and the query to be sent to it. It is then necessary to translate each query into the ontology of the resource 12, 14, 16 (step 94) and send this version to the wrapper 24, 26, 28 (step 96). On receiving a query, at step 98 the wrapper 24, 26, 28 translates it into the query language of the resource 12, 14, 16 retrieves the results of the query (step 100) and sends these results back to the query engine 30 (step 102). Each of the individual results then needs to be converted into the ontology of the query at step 104 before they can be integrated to give the results of the query as a whole. Once results to all of the sub-queries have been received and converted to the query ontology at step 104, the integration of those results begins. At step 106 each unexplored node in a solution graph is looped through. At step 108, each arc on the node is identified and the attached node retrieved, and at step 110 the linking attribute is retrieved. Once this is completed, as the graph has been compiled to provide an integrated solution to the query, this technique will ensure that all attributes and attribute conditions are retrieved, in effect by replacing each node with the result retrieved by the wrapper. The query engine can then compile the attributes in the appropriate format at step 112 and return this result to the query

source at step 114. An algorithm for dealing with this final step can be compiled in the manner adopted for the other stages discussed above.

It will be appreciated that variations of the system can be contemplated. Any number of resources of any database type or structure can be supported with the compilation of appropriate ontologies. Similarly any level of data or query structure, and network configuration or type can be used to implement the system, and the specific examples given in the description above are illustrative only.

## Claims

1. A database management system comprising a database manager and a plurality of database resources, in which the manager includes an ontology server for storing respective ontologies for each database resource.

2. A system as claimed in claim 1 in which the manager further includes a plurality of respective first stores each containing a mapping of a resource ontology to its respective resource contents.

3. A system as claimed in claim 1 or 2 in which the manager further includes a directory of the respective resource contents.

4. A system as claimed in claim 1 or 2 in which the manager further includes at least one second store containing mappings of the resource ontologies onto a pre-defined application ontology.

5. A system as claimed in any preceding claim in which the database manager comprises a query engine.

6. A method of managing a database comprising a database manager and a plurality of database resources comprising the steps of creating a resource ontology for each database resource and storing the resource ontology on the database manager.

7. A method as claimed in claim 6 further including the step of creating a mapping of each resource ontology to its respective resource contents, and storing the mapping in a respective first store.

8. A method as claimed in claim 7 further including the steps of generating queries to each of the first stores to obtain the respective resource contents and compiling therefrom a directory in the manager of the respective resource contents.

9. A method as claimed in any of claims 6 to 8 further including the steps of creating mappings of the resource ontologies onto a pre-defined application ontology and storing the mapping in at least one second store.

10. A computer readable medium comprising instructions for implementing a system as claimed in any of claims 1 to 5 and/or a method as claimed in any of claims 6 to 9.

Fig.1.

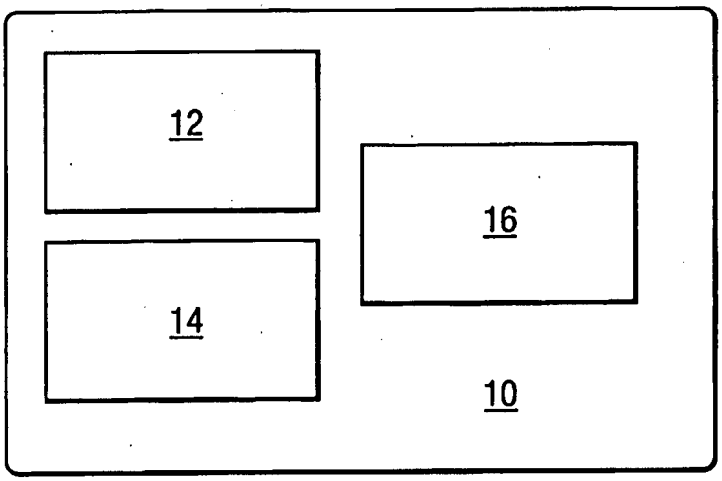

12
16
14
10

Fig.2.

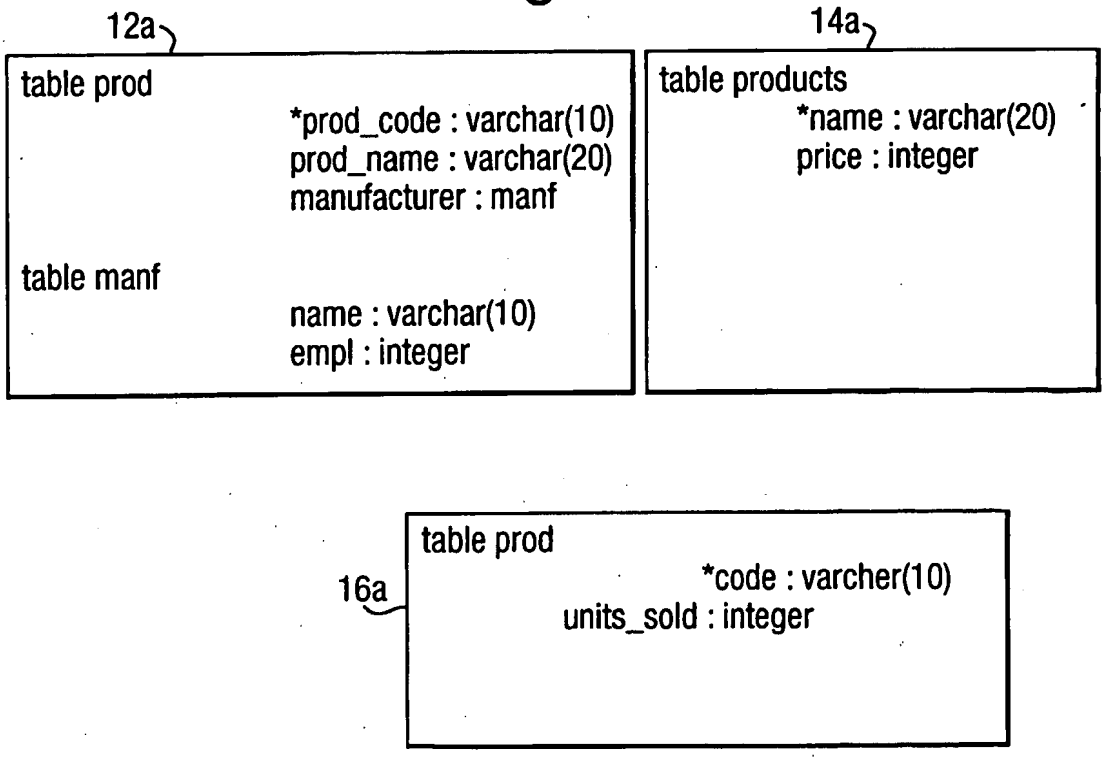

12a
table prod
*prod_code : varchar(10)
prod_name : varchar(20)
manufacturer : manf
table manf
name : varchar(10)
empl : integer
14a
table products
*name : varchar(20)
price : integer
16a
table prod
*code : varcher(10)
units_sold : integer

Fig.3.

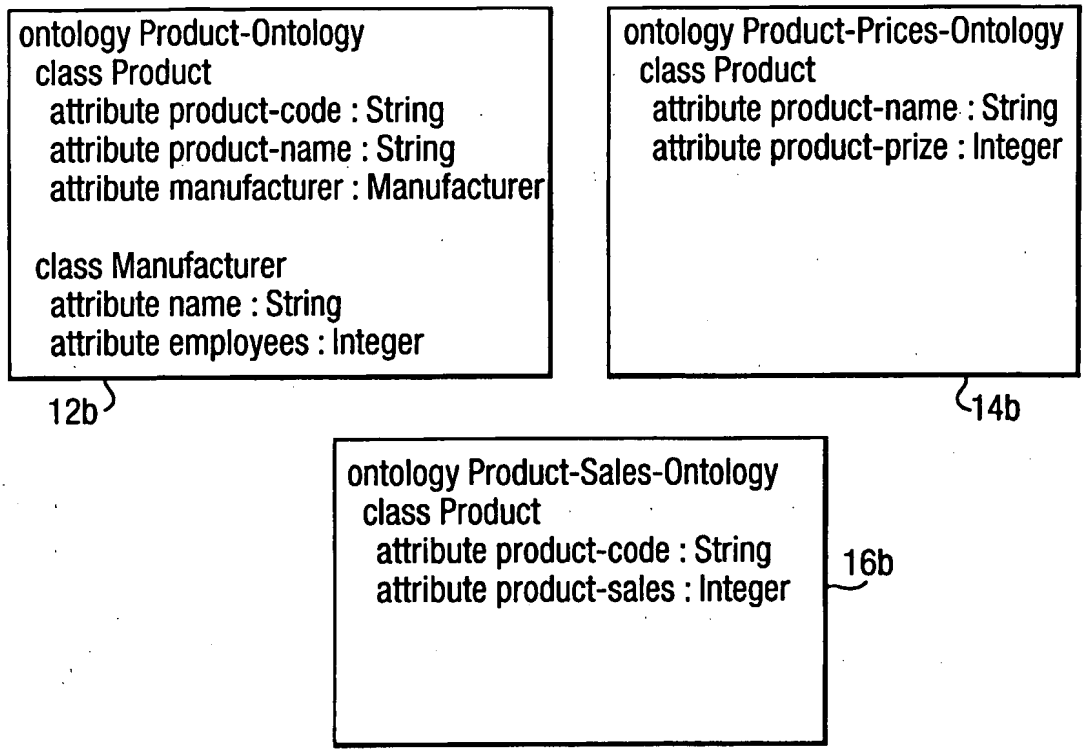

ontology Product-Ontology
class Product
attribute product-code : String
attribute product-name : String
attribute manufacturer : Manufacturer
class Manufacturer
attribute name : String
attribute employees : Integer
12b
ontology Product-Prices-Ontology
class Product
attribute product-name : String
attribute product-prize : Integer
14b
ontology Product-Sales-Ontology
class Product
attribute product-code : String
attribute product-sales : Integer
16b

Fig.4.

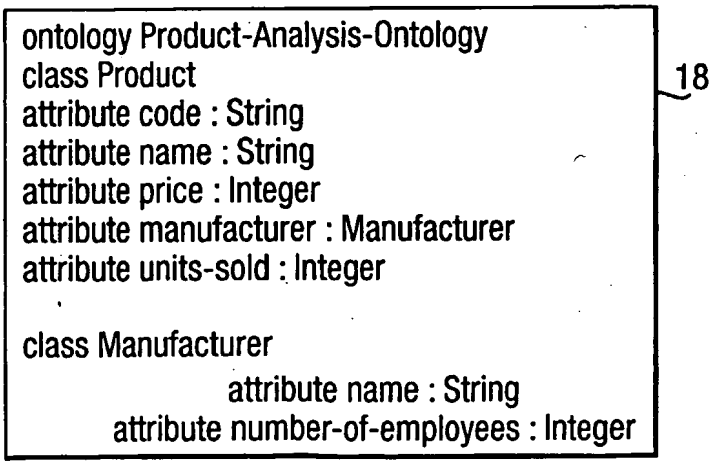

ontology Product-Analysis-Ontology
class Product
attribute code : String
attribute name : String
attribute price : Integer
attribute manufacturer : Manufacturer
attribute units-sold : Integer
class Manufacturer
attribute name : String
attribute number-of-employees : Integer
18

Fig.5.

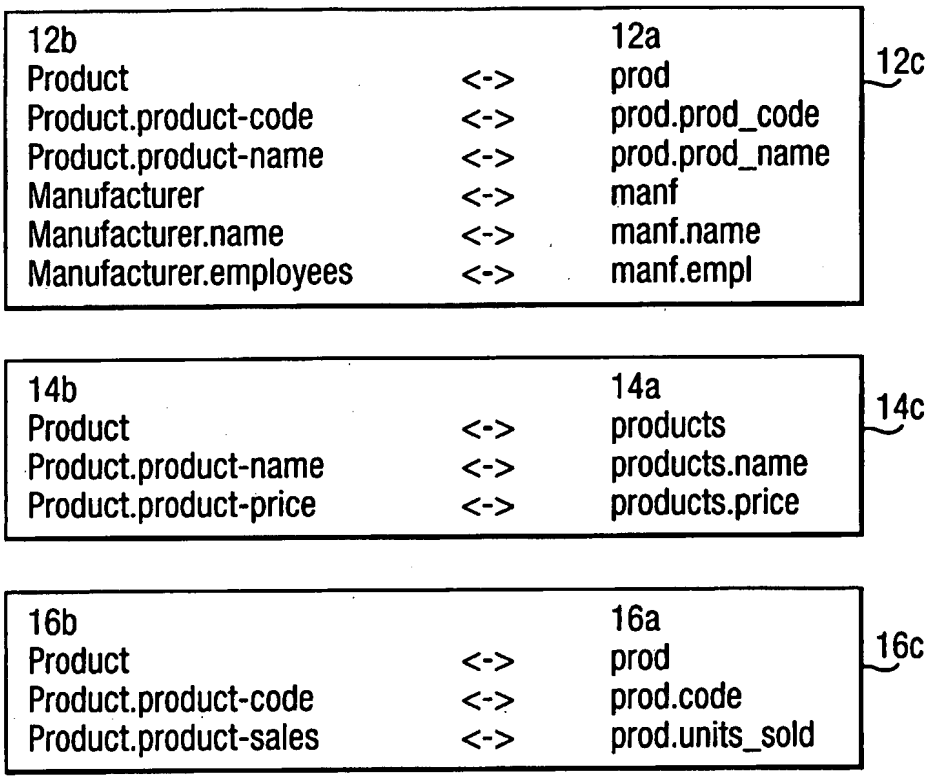

12b
12a
12c
Product <-> prod
Product.product-code <-> prod.prod_code
Product.product-name <-> prod.prod_name
Manufacturer <-> manf
Manufacturer.name <-> manf.name
Manufacturer.employees <-> manf.empl
14b
14a
14c
Product <-> products
Product.product-name <-> products.name
Product.product-price <-> products.price
16b
16a
16c
Product <-> prod
Product.product-code <-> prod.code
Product.product-sales <-> prod.units_sold

Fig.6.

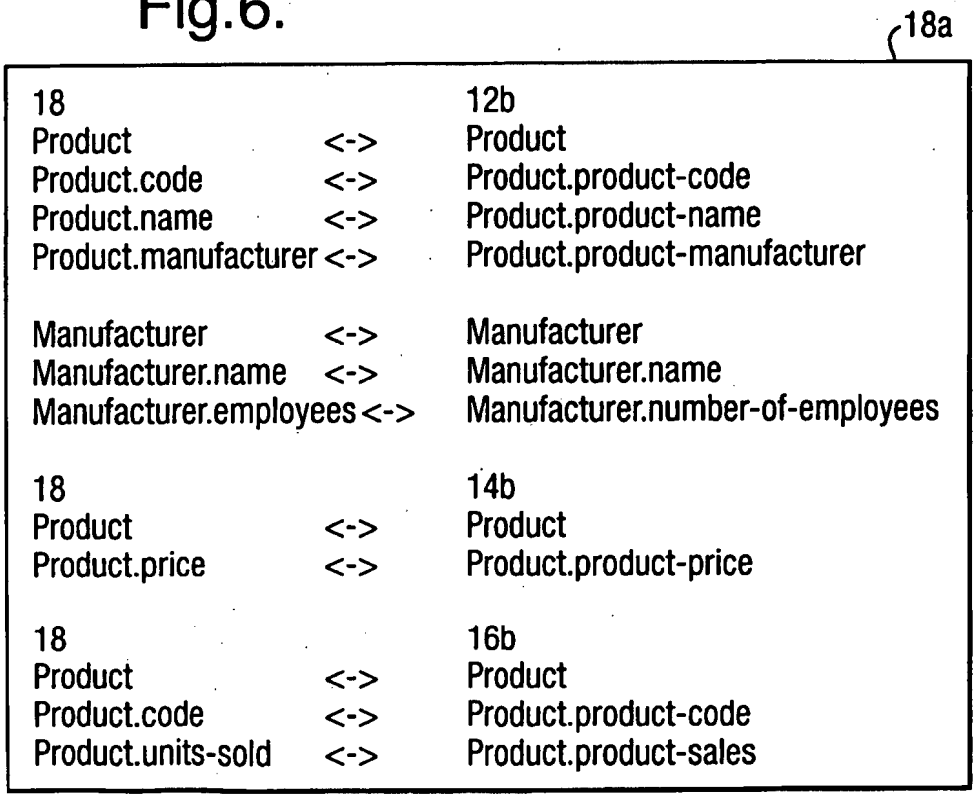

18a
18 12b
Product <-> Product
Product.code <-> Product.product-code
Product.name <-> Product.product-name
Product.manufacturer <-> Product.product-manufacturer
Manufacturer <-> Manufacturer
Manufacturer.name <-> Manufacturer.name
Manufacturer.employees <-> Manufacturer.number-of-employees
18 14b
Product <-> Product
Product.price <-> Product.product-price
18 16b
Product <-> Product
Product.code <-> Product.product-code
Product.units-sold <-> Product.product-sales

Fig.7.

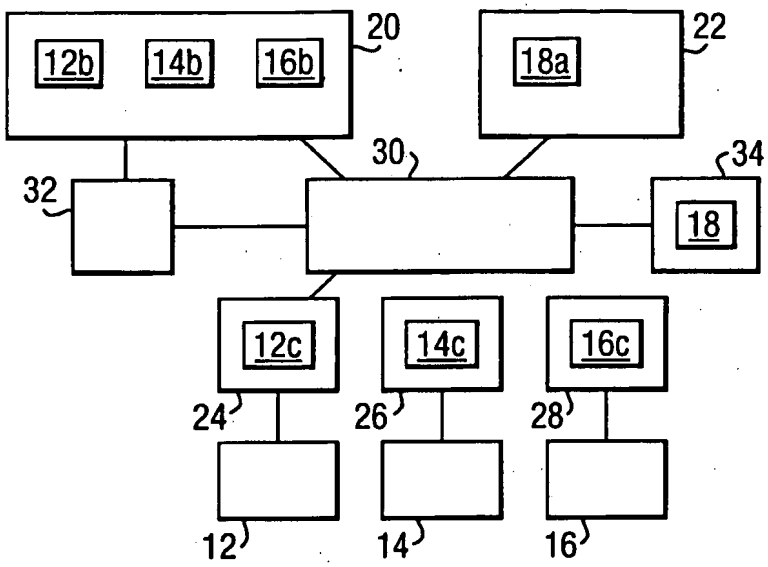

12b
14b
16b
20
18a
22
30
34
32
18
12c
14c
16c
24
26
28
12
14
16

Fig.8.

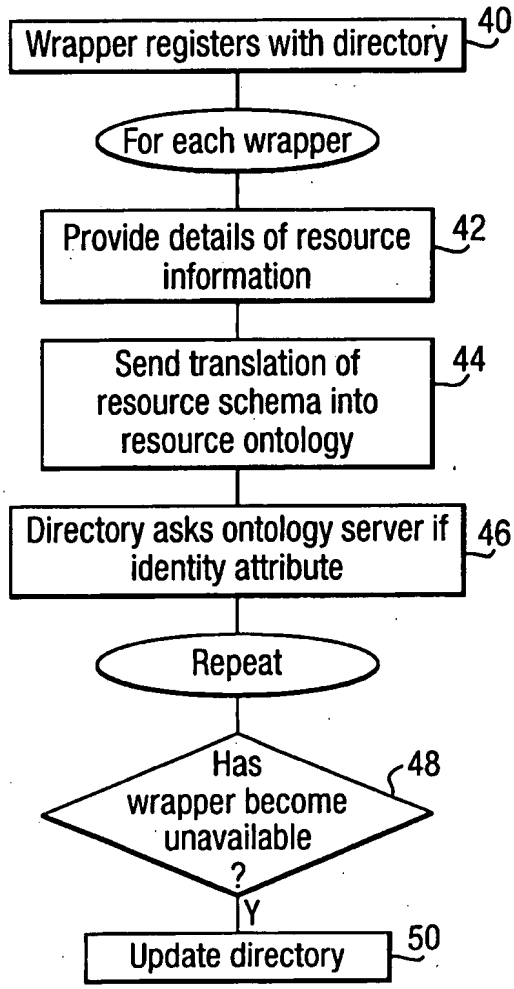

Wrapper registers with directory
40
For each wrapper
Provide details of resource information
42
Send translation of resource schema into resource ontology
44
Directory asks ontology server if identity attribute
46
Repeat
Has wrapper become unavailable ?
48
Y
Update directory
50

Fig.9.

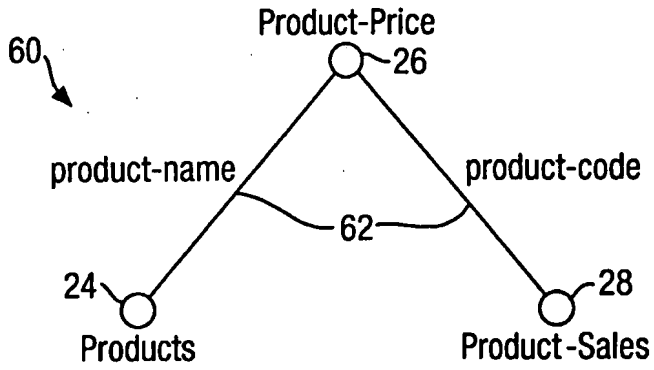

Product-Price
26
60
product-name
product-code
62
24
28
Products
Product-Sales

Fig.10.

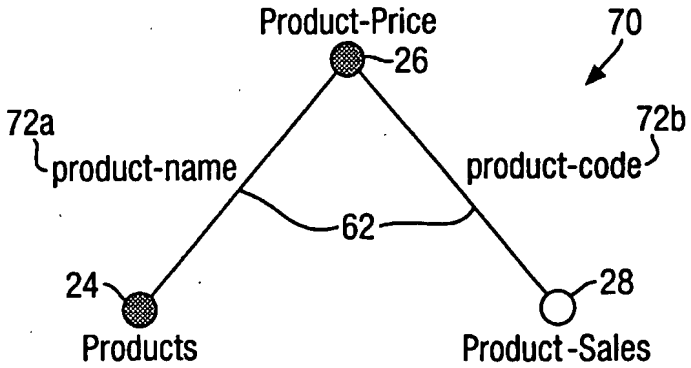

Product-Price
26
70
72a
72b
product-name
product-code
62
24
28
Products
Product-Sales

Fig.11.

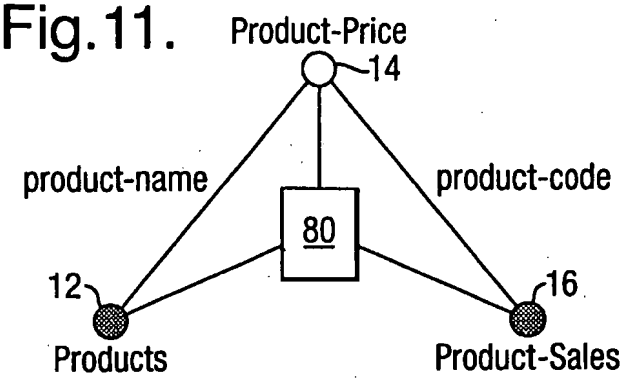
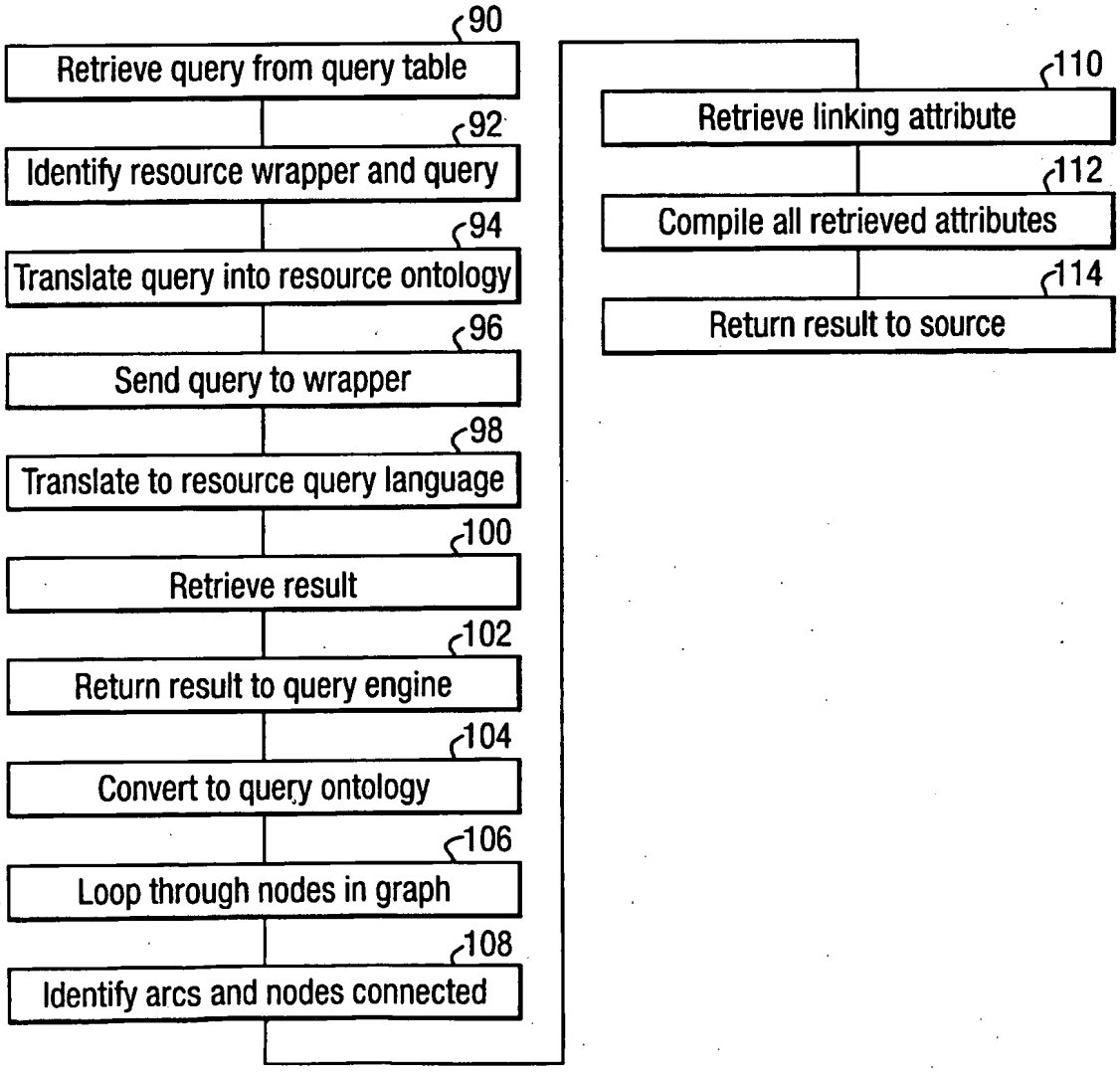

Product-Price
14
product-name
product-code
80
12
16
Products
Product-Sales

Fig.12.


90
Retrieve query from query table
92
Identify resource wrapper and query
94
Translate query into resource ontology
96
Send query to wrapper
98
Translate to resource query language
100
Retrieve result
102
Return result to query engine
104
Convert to query ontology
106
Loop through nodes in graph
108
Identify arcs and nodes connected
110
Retrieve linking attribute
112
Compile all retrieved attributes
114
Return result to source

# INTERNATIONAL SEARCH REPORT

Interna l Application No
PCT/GB 02/01231

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category ° | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | MENA, E. ET AL: "OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies" INTERNATIONAL JOURNAL ON DISTRIBUTED AND PARALLEL DATABASES (PAPD), KLUWER ACADEMIC PUBLISHERS, vol. 8, no. 2, April 2000 (2000-04), pages 1-50, XP002201997 Boston, USA Figures 4-6, sections 4,5 the whole document<br>---<br>-/-- | 1-10 |

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 13 June 2002 | 22/07/2002 |

| Name and mailing address of the ISA | Authorized officer |
|---|---|
| European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,<br>Fax: (+31-70) 340-3016 | Jaedicke, M |

2

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

| Category * | Citation of document, with indication,where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | MENA E ET AL: "Observer: an approach for query processing in global information systems based on interoperation across pre-existing ontologies" COOPERATIVE INFORMATION SYSTEMS, 1996. PROCEEDINGS., FIRST IFCIS INTERNATIONAL CONFERENCE ON BRUSSELS, BELGIUM 19-21 JUNE 1996, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC P, US, 19 June 1996 (1996-06-19), pages 14-25, XP010200745 ISBN: 0-8186-7505-5 page 16, left-hand column, paragraph 2.1 -page 22, right-hand column, paragraph 4 --- | 1-10 |
| X | VISSER, P.R.S. ET AL: "Resolving Ontological Heterogeneity in the KRAFT Project" PROCEEDINGS DATABASE AND EXPERT SYSTEMS APPLICATIONS, 10TH INTERNATIONAL CONFERENCE, DEXA-99 , 30 August 1999 (1999-08-30) - 3 September 1999 (1999-09-03), pages 668-677, XP002201998 Florence, Italy the whole document --- | 1-10 |
| A | GRAY P M D ET AL: "KRAFT: knowledge fusion from distributed databases and knowledge bases" DATABASE AND EXPERT SYSTEMS APPLICATIONS, 1997. PROCEEDINGS., EIGHTH INTERNATIONAL WORKSHOP ON TOULOUSE, FRANCE 1-2 SEPT. 1997, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, 1 September 1997 (1997-09-01), pages 682-691, XP010243362 ISBN: 0-8186-8147-0 the whole document --- | 1-10 |
| X | TAMER ÖZSU, M., VALDURIEZ, P.: "Principles of Distributed Database Systems" 1991 , PRENTICE-HALL INTERNATIONAL , USA XP002201999 page 74, line 1 -page 93, line 7 --- -/-- | 1-10 |

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

| Category * | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | CAREY M J ET AL: "Towards heterogeneous multimedia information systems: the Garlic approach" RESEARCH ISSUES IN DATA ENGINEERING, 1995: DISTRIBUTED OBJECT MANAGEMENT, PROCEEDINGS. RIDE-DOM '95. FIFTH INTERNATIONAL WORKSHOP ON TAIPEI, TAIWAN 6-7 MARCH 1995, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, 6 March 1995 (1995-03-06), pages 124-131, XP010129045 ISBN: 0-8186-7056-8 page 126, left-hand column, last paragraph -page 130, right-hand column, last paragraph; figure 1 --- | 1-10 |
| X | ROTH M T ET AL: "DON'T SCRAP IT WRAP IT: A WRAPPER ARCHITECTURE FOR LEGACY DATA SOURCES" PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB97), 26 August 1997 (1997-08-26), pages 266-275, XP000940797 the whole document --- | 1-10 |
| X | BAYARDO JR R J ET AL: "INFOSLEUTH: AGENT-BASED SEMANTIC INTEGRATION OF INFORMATION IN OPEN AND DYNAMIC ENVIRONMENTS" SIGMOD RECORD, ASSOCIATION FOR COMPUTING MACHINERY, NEW YORK, US, vol. 26, no. 2, 1 June 1997 (1997-06-01), pages 195-206, XP000730507 the whole document --- | 1-10 |
| X | NODINE M ET AL: "Semantic brokering over dynamic heterogeneous data sources in InfoSleuth" DATA ENGINEERING, 1999. PROCEEDINGS., 15TH INTERNATIONAL CONFERENCE ON SYDNEY, NSW, AUSTRALIA 23-26 MARCH 1999, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, 23 March 1999 (1999-03-23), pages 358-365, XP010326167 ISBN: 0-7695-0071-4 the whole document --- | 1-10 |
| X | WO 00 65486 A (WONG BENSON T; KENDALL ELISA F (US); LAI ERIC (US); WONG JAMES S () 2 November 2000 (2000-11-02) abstract; figures 4-11 --- -/-- | 1-10 |

2

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

| Category ° | Citation of document, with indication,where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X | US 5 634 053 A (NOBLE WILLIAM B ET AL) 27 May 1997 (1997-05-27) the whole document | 1-10 |
| A | US 5 970 490 A (MORGENSTERN MATTHEW) 19 October 1999 (1999-10-19) the whole document | 1-10 |
| A | EP 0 829 811 A (NIPPON TELEGRAPH & TELEPHONE) 18 March 1998 (1998-03-18) the whole document | 1-10 |
| A | US 5 590 319 A (COHEN GERALD D ET AL) 31 December 1996 (1996-12-31) the whole document | 1-10 |

# INTERNATIONAL SEARCH REPORT

Information on patent family members

Interna ll Application No
PCT/GB 02/01231

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| WO 0065486 A | 02-11-2000 | AU 6334000 A | 10-11-2000 |
| | | WO 0065486 A2 | 02-11-2000 |
| US 5634053 A | 27-05-1997 | NONE | |
| US 5970490 A | 19-10-1999 | NONE | |
| EP 0829811 A | 18-03-1998 | EP 0829811 A1 | 18-03-1998 |
| | | JP 10143539 A | 29-05-1998 |
| | | US 6233578 B1 | 15-05-2001 |
| US 5590319 A | 31-12-1996 | NONE | |